Trabalho prático - AEDs III 2021-1/UFSJ

Prof. Alexandre Bittencourt Pigozzo.
Valor do trabalho: 40 pontos.
Data de entrega: **09/08/2021** (segunda-feira) até às 23:59.

O trabalho prático pode ser feito **individualmente ou em dupla**. Comece a fazer o trabalho o quanto antes. Entregar através do portal didático todo o código e documentação compactados em um arquivo com o nome do aluno ou da dupla. O trabalho prático deverá ser apresentado pelo aluno ou dupla. A apresentação deverá ser agendada com o professor entre os dias 10/08/2021 e 13/08/2021. O dia e horário da apresentação dependerá da disponibilidade de horário do professor. O trabalho pode ser feito em qualquer linguagem de programação (ver observação a seguir).

Obs: Favor implementar o trabalho nas linguagens de programação mais tradicionais e conhecidas como C, C++, Python, Java, Javascript, etc. Caso você queira implementar em uma linguagem não tão conhecida, favor enviar todas as bibliotecas e instruções de instalação para executar o programa no sistema operacional Ubuntu 20.04. Poderá ser atribuída nota zero ao trabalho caso o professor não consiga executar o trabalho no Ubuntu 20.04.

O objetivo deste trabalho é aprender na prática a trabalhar com a estrutura de dados do tipo Grafo e aplicá-la em problemas reais da área de Compiladores.

Uma das fases mais importantes da compilação é a fase de geração de código e otimização. Essa fase é responsável por gerar o código Assembly (código alvo) e realizar diversas otimizações durante a geração do código Assembly com o objetivo de gerar o código mais eficiente possível.

Para auxiliar a fase de geração de código e otimização, uma representação muito utilizada nos compiladores é chamada código de três endereços. A partir do código de três endereços, podemos construir um **Grafo de Fluxo de Controle (GFC)** que nos dará várias informações importantes sobre o fluxo de controle do código e esse grafo será a base para várias otimizações que são realizadas no compilador.

A ideia do trabalho prático é criar um GFC a partir de um código de três endereços dado como entrada para o programa. O trabalho será explicado em mais detalhes nas próximas seções, onde primeiro são apresentados os conceitos básicos para entendê-lo.

Código de três endereços

O código de três endereços é uma representação intermediária (RI) usada no back-end do compilador para ajudar a gerar código de máquina. É uma RI interessante porque está bem próxima da linguagem Assembly. No código de três endereços, existe no máximo **um operador** no lado direito de uma instrução e no máximo **três endereços** na instrução. Por exemplo, uma expressão x + y * z, no código de três endereços é traduzida para:

```
t1 = y * z
t2 = x + t1
```

Onde t1 e t2 são temporários gerados pelo compilador.

Obs: Em cada instrução de três endereços, um endereço pode representar o endereço de um operando fonte, ou de um operando destino ou o nome de um label que é destino de um desvio condicional.

No código de três endereços, um endereço pode ser:

- Um nome: Por conveniência, permitimos que os nomes de identificadores do programa fonte aparecem como endereços no código de três endereços. Em uma implementação, o identicador é substituído por um ponteiro para sua entrada na tabela de símbolos.
- Uma constante: Na prática, um compilador deve tratar muitos tipos diferentes de constantes, fazendo as conversões de tipo onde for permitido pela hierarquia de tipos.
- Um temporário gerado pelo compilador: é vantajoso, especialmente em compiladores otimizados, criar um nome distinto toda vez que um temporário é necessário.

Uma possível especificação de uma linguagem de três endereços envolve quatro tipos básicos de instruções: atribuição, desvios, invocação de rotinas e acesso indexado e indireto.

Instruções de atribuição são aquelas nas quais o resultado de uma operação é armazenado na variável especificada à esquerda do operador de atribuição, aqui denotado por =. Há três formas para esse tipo de instrução.
Na primeira, a variável recebe o resultado de uma operação binária:
x = y op z
O resultado pode ser também obtido a partir da aplicação de um operador unário:
x = op y
Na terceira forma, pode ocorrer uma simples cópia de valores de uma variável para outra:
x = y
Por exemplo, a expressão em C

a = b + c * d;
seria traduzida nesse formato para as instruções:

```
t1 = c * d
t2 = b + t1
a = t2
```

onde t1 e t2 são temporários criados pelo compilador.

Obs: O resultado da expressão no lado direito é primeiro colocado em um temporário para depois ser copiado para a variável do lado esquerdo. Isso ocorre por dois motivos:

- Temos que introduzir uma lógica adicional no gerador de código para ele saber se está gerando código para a última expressão a ser calculada no lado direito e com isso colocar o resultado dessa expressão na variável;
- Colocar os resultados das expressões em temporários introduz a possibilidade de realizar otimizações como, por exemplo, se o valor de t2 será utilizado mais para frente ele pode ser mantido em um registrador gerando um código mais otimizado.

As instruções de desvio podem assumir duas formas básicas. Uma instrução de **desvio incondicional** tem o formato
goto L
onde L é um rótulo simbólico que identifica uma linha do código. A outra forma de desvio é o **desvio condicional**, com o formato
if x opr y goto L
onde opr é um operador relacional de comparação e L é o rótulo da linha que deve ser executada se o resultado da aplicação do operador relacional for verdadeiro; caso contrário, a linha seguinte é executada.
Por exemplo, a seguinte iteração em C

```
while (i++ <= k)
  x[i] = 0;
x[0] = 0;
```

poderia ser traduzida para

```
L1: if i > k goto L2
    t1 = i + 1
    i = t1
    x[i] = 0
    goto L1
L2: x[0] = 0
```

A invocação de rotinas ocorre em duas etapas. Inicialmente, os argumentos do procedimento são ``registrados'' com a instrução param; após a definição dos argumentos, a instrução call completa a invocação da rotina. A instrução return indica o fim de execução de uma rotina. Opcionalmente, esta instrução pode especificar um valor de retorno, que pode ser atribuído na linguagem intermediária a uma variável como resultado de call.

Por exemplo, considere a chamada de uma função f que recebe três argumentos e retorna um valor:
f(a, b, c);

Neste exemplo em C, esse valor de retorno não é utilizado. De qualquer modo, a expressão acima seria traduzida para

```
param a
param b
param c
t1 = call f,3
```

onde o número após a vírgula indica o número de argumentos utilizados pelo procedimento f. Com o uso desse argumento adicional é possível expressar sem dificuldades as chamadas aninhadas de procedimentos.

Obs: Para este trabalho, será considerado que a instrução call não muda o fluxo de controle/execução do código.

O último tipo de instrução para códigos de três endereços refere-se aos modos de endereçamento indexado e indireto. Para atribuições indexadas, as duas formas básicas são

```
x = y[i]
x[i] = y
```

As atribuições associadas ao modo indireto permitem a manipulação de endereços e seus conteúdos. As instruções em formato intermediário também utilizam um formato próximo àquele da linguagem C:

```
x = &y
w = *x
*x = z
```

A tabela abaixo resume as instruções de três endereços mais comuns:

Formato	Instrução
0	z = x
1	z = x op y
2	if x oprel y goto L
3	ifnot x oprel y goto L
4	goto L
5	param xi
6	call p, n

p = nome do procedimento chamado
n = número de parâmetros passados

Uma representação mais simples das instruções em códigos de três endereços dá-se na forma de armazenamento em tabelas com quatro ou três colunas. Na abordagem que utiliza quádruplas (as tabelas com quatro colunas), cada instrução é representada por uma linha na tabela com a especificação do operador, do primeiro argumento, do segundo argumento e do resultado. Por exemplo, a tradução do código abaixo, resultaria no seguinte trecho da tabela:

Tabela para o código dado abaixo:

inst	operador	arg1	arg2	resultado
1	*	c	d	t1
2	+	b	t1	t2
3		t2		a
4	<	a	b	L1

```
t1 = c * d
t2 = b + t1
a = t2
if a < b goto L1
```

Obs: Podemos armazenar um campo para o tipo da instrução também.

Obs: Para algumas instruções, como aquelas envolvendo comandos de cópia, operadores unários ou desvio incondicional, algumas das colunas estariam vazias.

Exemplos de códigos de três endereços

Abaixo é dado uma tabela que mostra exemplos de códigos de três endereços gerado para diferentes entradas. Para algumas entradas, só é mostrado o trecho de um código ou uma

expressão para facilitar o entendimento.

Entrada	Código de três endereços

int x, y, x2, y2;
x2 = x*x;
y2 = y*y;
x2 = x2 + y2;
	t0 = x * x
	x2 = t0
	t1 = y * y
	y2 = t1
	t2 = t0 + t1
	x2 = t2

a + a*(b - c) + (b - c)*d	t1 = b - c t2 = a * t1 t3 = b - c t4 = t3 * d t5 = a + t1 t6 = t5 + t4

int z;
if (2 || 4)
  z = 1;
else
  z = 0;
	if 2 != 0 goto L1
	if 4 == 0 goto L2
	goto L1
	L1: z = 1
	goto L0
	L2: z = 0
	L0:

int x = 0, y = 1, z;
if (x && y)
  z = 1;
else
  z = 0;
	x = 0
	y = 1
	if x == 0 goto L2
	if y == 0 goto L2
	goto L1
	L1: z = 1
	goto L0
	L2: z = 0
	L0:

int x = 0, y = 1, z;
if (x || y)
  z = 1;
else
  z = 0;
	x = 0
	y = 1
	if x != 0 goto L1
	if y == 0 goto L2
	L1: z = 1
	goto L0
	L2: z = 0
	L0:

int x = 0, y = 1, z;
x = 3 + 2 - x;
if (x && y || z)
  z = 1;
else
  z = 0;
	x = 0
	y = 1
	t0 = 3 + 2
	t1 = t0 - x
	x = t1
	if x == 0 goto L3
	if y == 0 goto L3
	goto L1
	L3: if z == 0 goto L2
	goto L1
	L1: z = 1
	goto L0
	L2: z = 0
	L0:

Entrada	Código de três endereços

int x = 0, y = 1, z;
x = 3 + 2 - x;
if (x < y && x >= y)
  z = 1;
else
  z = 0;
	x = 0
	y = 1
	t0 = 3 + 2
	t1 = t0 - x
	x = t1
	if x < y goto L3
	goto L2
	L3: if x >= y goto L1
	goto L2
	L1: z = 1
	goto L0
	L2: z = 0
	L0:

int a = 1, b = 2, c = 3; a = b < c;	a = 1 b = 2 c = 3 if b < c goto L0 t0 = 0 goto L1 L0: t0 = 1 L1: a = t0
int a = 1, b = 2, c = 3; a = b || a + c;	a = 1 b = 2 c = 3 t1 = a + c if b != 0 goto L0 if t1 != 0 goto L0 t0 = 0 goto L1 L0: t0 = 1 L1: a = t0

int a = 1, b = 2, c = 3;
while (c > 0) {
  a = a + 1;
  c = c - 1;
}
	a = 1
	b = 2
	c = 3
	L0: if c>0 goto L1
	goto L2
	L1: t0 = a + 1
	a = t0
	t1 = c - 1
	c = t1
	goto L0
	L2:

int a = 1, b = 2, c = 3;
while (c != 5 && a < 2 || b == 1) {
  a = a + 1;
  c = c - 1;
}
	a = 1
	b = 2
	c = 3
	L0: if c!=5 goto L4
	goto L3
	L4: if a<2 goto L1
	goto L3
	L3: if b==1 goto L1
	goto L2
	L1: t0 = a + 1
	a = t0
	t1 = c - 1
	c = t1
	goto L0
	L2:

Blocos Básicos e Grafos de Fluxo de Controle (GFCs)

Para realizar diversas otimizações no código, um compilador constrói grafos de fluxo de controle (GFCs) a partir das intruções de três endereços. Um GFC é um grafo onde os nós são blocos básicos e a ligação entre os nós representa a passagem do fluxo de controle de um bloco para outro. A seguir é dada a definição de bloco básico.

Bloco básico: sequência de instruções de três endereços com as seguintes propriedades:
(a) O fluxo de controle só pode entrar no bloco básico por meio da primeira instrução do bloco.
(b) O fluxo de controle sairá do bloco somente na última instrução.

Passos para a construção de um GFC:
1. Particione o código intermediário em blocos básicos;
2. Os blocos básicos formam os nós de um GFC e as ligações entre os blocos são as arestas do GFC.

Obs: Código intermediário é o código de três endereços.

Algoritmo (Particionamento de instruções de três endereços em blocos básicos):
Entrada: Sequência de instruções de três endereços.
Saída: Uma lista de blocos básicos para essa sequência.
Método:
Primeiro determine as instruções que são **líderes**:

1. A primeira instrução no código intermediário é um líder.
2. Qualquer instrução que seja o destino de um desvio condicional ou incondicional é um líder.
3. Qualquer instrução que segue imediatamente um desvio condicional é um líder.

Depois, os blocos básicos são construídos da seguinte forma:

1. O primeiro bloco básico vai da instrução que é o primeiro líder até a instrução anterior ao próximo líder;
2. O segundo bloco básico vai da instrução que é o segundo líder até a instrução anterior ao próximo líder;
3. O terceiro bloco básico vai do terceiro líder até a instrução anterior ao próximo líder.
4. E, assim por diante.

Grafos de fluxo

Uma vez que um programa em código intermediário (código de três endereços) é particionado em blocos básicos, representamos o fluxo de controle entre eles por meio de um GFC. Os nós de um grafo de fluxo são os blocos básicos. Existe uma aresta do bloco B para o bloco C se e somente se for possível que o primeiro comando no bloco C venha imediatamente após o último comando no bloco B. Existem duas maneiras pelas quais esse tipo de aresta poderia ser justificada:

• **Existe um desvio condicional ou incondicional a partir do fim de B para o início de C**.

• **C vem imediatamente após B na ordem original dos comandos de três endereços, e B não termina com um desvio incondicional**.

Dizemos que B é um predecessor de C, e C é um sucessor de B.

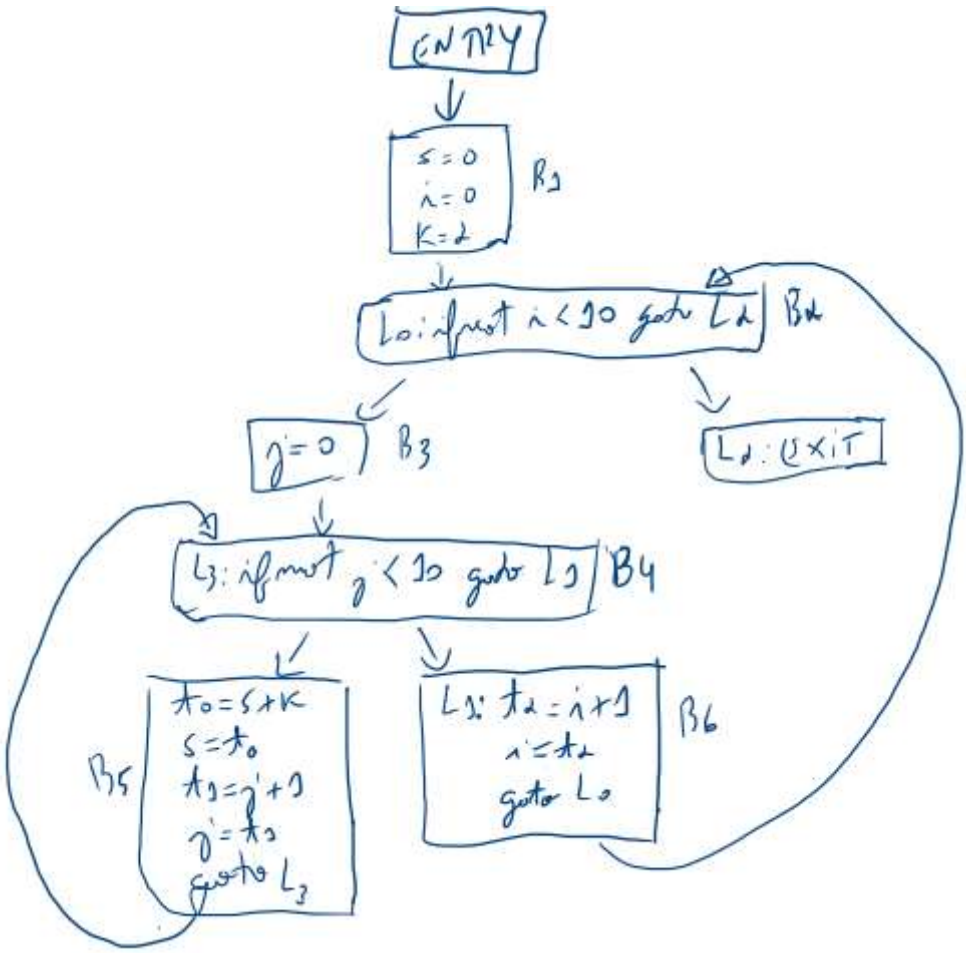
Frenquentemente, incluímos no grafo de fluxo dois nós, chamados de entrada e saída, que não fazem parte das instruções intermediárias executáveis. Existe uma aresta da entrada para o primeiro nó executável do grafo de fluxo. Há uma aresta para a saída a partir de um ou mais blocos básicos que contém as últimas instruções a serem executadas.

A seguir, são dados dois exemplos de GFCs construídos para dois códigos de três endereços de entrada.

Código de três endereços de entrada:

```
    s = 0
    i = 0
    k = 2
L0: ifnot i < 10 goto L2
    j = 0
L3: ifnot j < 10 goto L1
    t0 = s + k
    s = t0
    t1 = j + 1
    j = t1
    goto L3
L1: t2 = i + 1
    i = t2
    goto L0
L2:
```

Grafo de fluxo para o código acima:



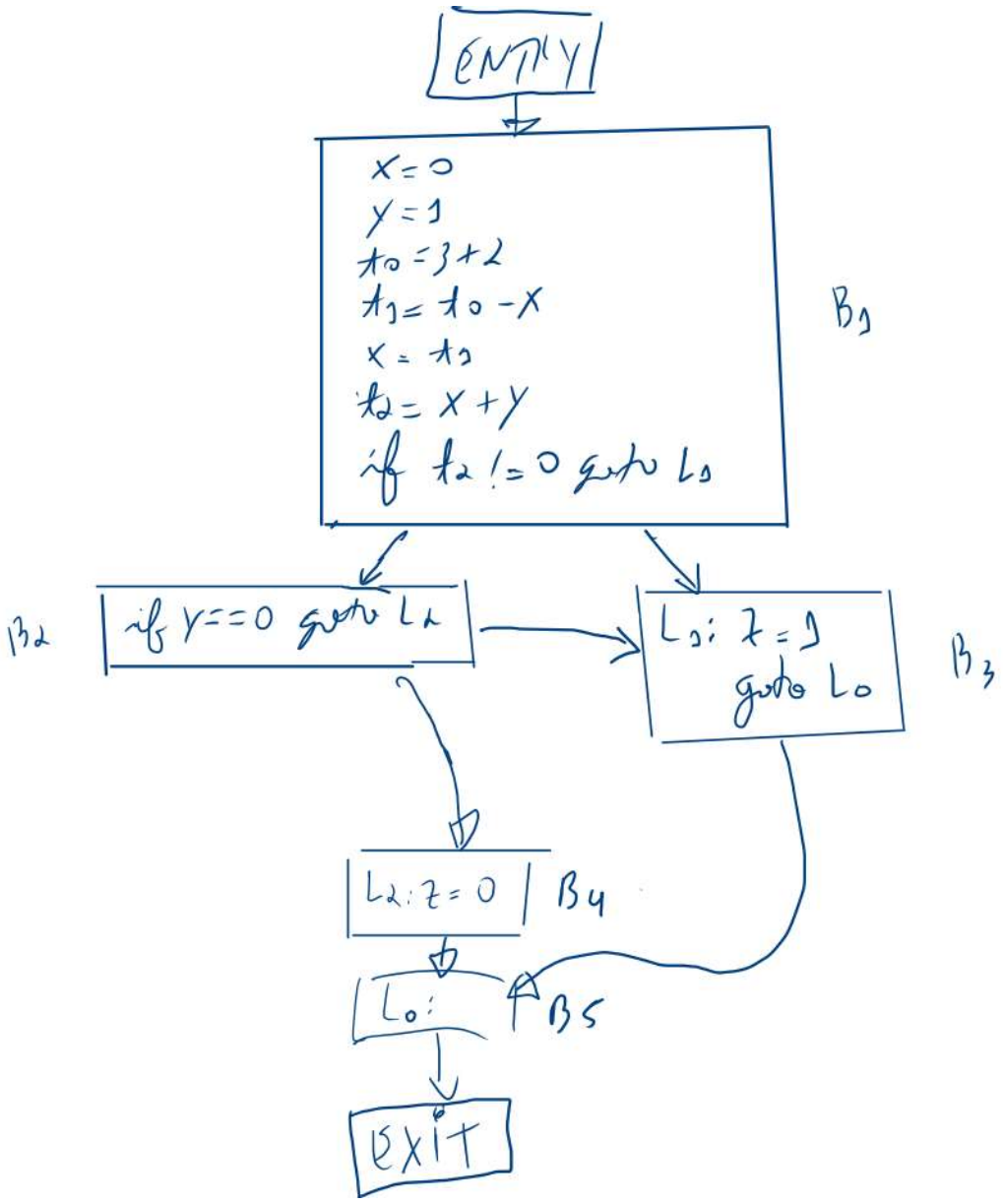
Os blocos básicos (nós do grafo) estão nomeados como B1, B2, …, até B6. As ligações entre os blocos básicos representam as arestas do grafo.

Outro exemplo é dado a seguir.

Código de três endereços de entrada:

```
    x = 0
    y = 1
    t0 = 3 + 2
    t1 = t0 - x
    x = t1
    t2 = x + y
    if t2 != 0 goto L1
    if y == 0 goto L2
L1: z = 1
    goto L0
L2: z = 0
L0:
```

GFC para o código acima:



Os blocos básicos (nós do grafo) são nomeados com a letra B seguida por um número. As ligações entre os blocos básicos representam as arestas do grafo.

O que fazer

Com base na descrição anterior, neste trabalho prático vocês devem implementar um programa que recebe como entrada um arquivo de texto contendo um código de três endereços (uma instrução por linha) e a partir desse código o programa exibe como saída o Grafo de Fluxo de Controle (GFC) criado para o código de entrada. O programa deve imprimir todos os blocos básicos mostrando o conteúdo de cada um e imprimir as ligações entre os blocos básicos. Opcionalmente, podem ser definidos também os nós especiais de entrada (chamado Entry) e saída (chamado Exit). Caso o bloco básico do final do código tenha somente um label, ele pode ser substituído pelo bloco especial Exit.

Obs: Dentro da condição da instrução if só aparecem operadores relacionais. À direita de um comando de atribuição, só aparecem cópias ou operações com os operadores aritméticos.

A seguir, são dados dois exemplos de entrada e as saídas correspondentes.

Entrada:

```
x = 0
y = 1
t0 = 3 + 2
t1 = t0 - x
x = t1
t2 = x + y
if t2 != 0 goto L1
if y == 0 goto L2
L1: z = 1
goto L0
L2: z = 0
L0:
```

Saída:

```
B1: x = 0
y = 1
t0 = 3 + 2
t1 = t0 - x
x = t1
t2 = x + y
if t2 != 0 goto L1

B2: if y == 0 goto L2

B3: L1: z = 1
goto L0

B4: L2: z = 0

B5: L0:

Ligações:
Entry -> B1
B1 -> B2, B3
B2 -> B3, B4
B3 -> B5
```

B4 -> B5
B5 -> Exit

Entrada:

s = 0
i = 0
k = 2
L0: ifnot i < 10 goto L2
j = 0
L3: ifnot j < 10 goto L1
t0 = s + k
s = t0
t1 = j + 1
j = t1
goto L3
L1: t2 = i + 1
i = t2
goto L0
L2:

Saída:

B1:
s = 0
i = 0
k = 2

B2:
L0: ifnot i < 10 goto L2

B3:
j = 0

B4:
L3: ifnot j < 10 goto L1

B5:
t0 = s + k
s = t0
t1 = j + 1
j = t1
goto L3

B6:
L1: t2 = i + 1
i = t2
goto L0

Exit:
L2

Ligações:
Entry -> B1
B1 -> B2
B2 -> B3, Exit
B3 -> B4
B4 -> B5, B6
B5 -> B4
B6 -> B2

Obs: A instrução ifnot tem a mesma sintaxe da instrução if tendo como condição uma expressão relacional e um label de destino para o desvio.

O que deve ser entregue

Ao final, devem ser entregues os seguintes documentos:

- Código fonte bem organizado, identado e comentado.
- Documentação descrevendo como a estrutura de dados do grafo foi implementada, como a entrada foi processada, como foi feita a lógica para criar o GFC, entre outras informações que julgar relevante.
- Resultados de cinco testes diferentes (ver os códigos de exemplo da tabela desta especificação) respondendo para cada teste as seguintes questões:
  - A partir do GFC construído, quais são os possíveis fluxos de execução do programa?
  - Quais são os loops no GFC caso existam?
- No final do documento descrever as dificuldades encontradas na realização do trabalho, o que foi implementado e o que não foi implementado.

O que será avaliado

Funcionamento correto do programa, uso correto das estruturas de dados, uso correto da orientação a objetos (OO) caso seja implementado com OO, código organizado, bem

identado, comentado, texto da documentação e os resultados. Além disso, também será avaliada a apresentação do trabalho que é obrigatória. Durante a apresentação, poderão ser feitas diversas perguntas ao aluno ou aos membros da dupla.

Observações

Todos os códigos são vistos e analisados linha por linha e todos eles são submetidos a uma ferramenta anti-plágio para avaliar a semelhança entre os códigos.